Mini Search Engine

Author : Hao Liu

Group 11:
Hao Liu
Qilan Ren
Shuai Yan

Date: 2019-03-20

Chapter 1: Introduction

In this project, we will create our own mini search engine which can handle inquiries over "The Complete Works of William Shakespeare" (http://shakespeare.mit.edu/).

We have done:

(1) Run a word count over the Shakespeare set and try to identify the stop words.

(2) Create our own inverted index over the Shakespeare set with word stemming.

(3) Write a search engine with the inverted index we built in step 2, which can return the file names that contain the key word.

(4) Run tests to show how the thresholds on query may affect the results.

First we use WEB crawler to download "The Complete Works of William Shakespeare" and store the text files in the directory named ShakespeareComplete. Then we build a mini search engine on the top of these files.

The whole project is divided into two parts. The **pretreater** and the **search engine**. (They both work well on Mac OS, if you want to run them on Windows, you may have to rewrite some part of the code.)

Pretreater:

(1) Runs a word count over the Shakespeeare set and identify the stop words then store them in the text file named StopWordsList.

(2) Stem all the words of all the text files and store the stemmed text files in the directory named Stemmed.

(3) Output all the file names in the text file named Filenames List.

Search Engine:

(1) Create the inverted index over the stemmed text files.

(2) Accept a key word and output all the file names that contain the word.

Chapter 2: Data Structure / Algorithm Specification

2.1 Pretreater

int main()

```
Create the directory Stemmed;
Open every text file in ShakespeareComplete{
   Read every word and stem it;
   Record the frequency of each word’s appearance;
   Output each file’s name in FilenamesList.txt;
   Output the stemmed text in the directory Stemmed;
}
Sort the words by their frequency decreasingly;
Select the words whose frequency is larger than a certain value, output them in StopWordsList.txt;
```

2.2 Search Engine

```
1 typedef vector<pair<string, int>> postList;//文件名，出现次数
2
3 class searchEngine{
```

```
 4 private:
 5     set<string> stopWords;//包含所有 stop words 的集合
 6     vector<string> filenames;
 7     string dirname = "Stemmed";//存放文本的子目录名
 8     map<string, postList*> invertedIndex;//搜索时使用的倒排文
件索引
 9
10 public:
11     void loadStopWords();//加载 stop words 到集合中
12     void loadFilenames();//加载文本文件名到 vector 中
13     void initialIndex();//初始化倒排文件索引
14     vector<string> search(string& keyWords, double threshold
= 1.0);//搜索，输入关键词和 threshold，返回包含关键词的文件名列表
15 };
```

void loadStopWords()

Read StopWordsList.txt;

Put every stop word in the set<string> stopWords;

void loadFilenames()

Read FilenamesList.txt;

Put every file name in the vector<string> filenames;

void initialIndex()

Open every file in directory Stemmed{

   Read every word;

   if it’s a new word

      then insert a new term in map<string, postList*>
invertedIndex;

   else update the corresponding postList;

}

Output the size of the invertedIndex;

vector<string> search(string& keyWords, double threshold = 1.0)

Stem the keyWord;

Find the corresponding term of keyWord in invertedIndex;

if there is at least one term

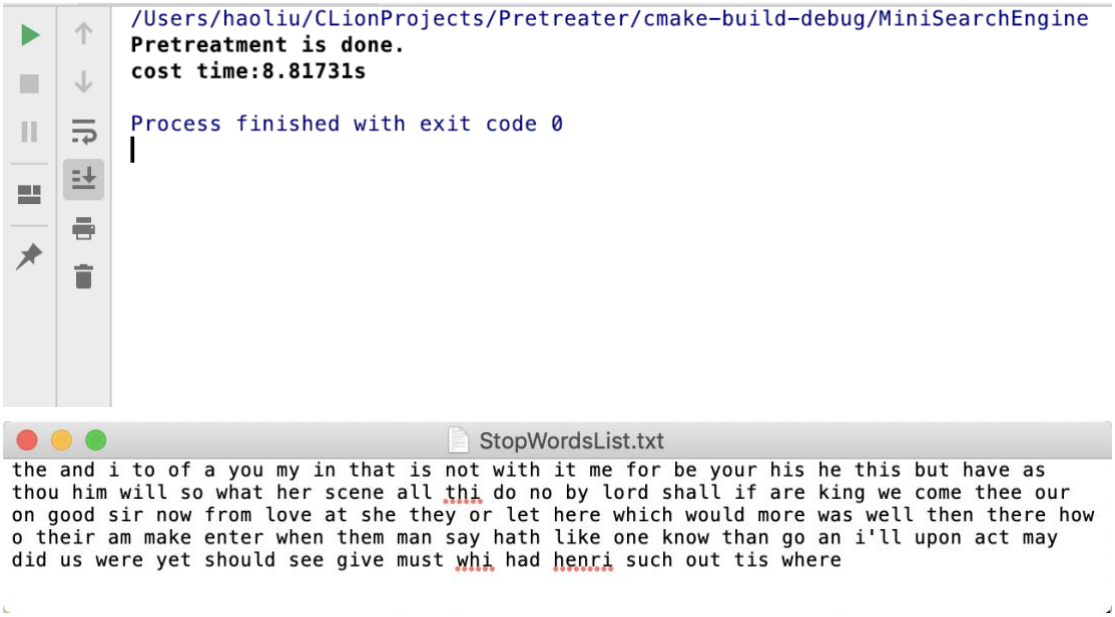
then sort the terms by frequency of the keyWord's appearence;

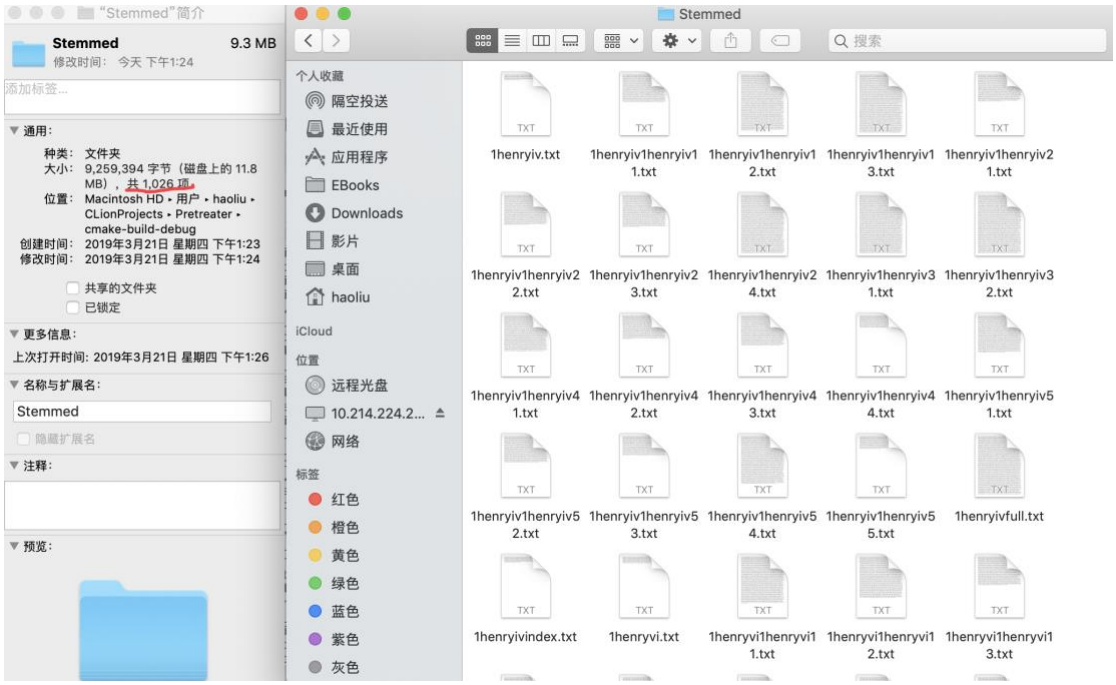
Put every file's name in the vector<string> filenames;

return filenames;

Chapter 3: Testing Results

3.1 Pretreater

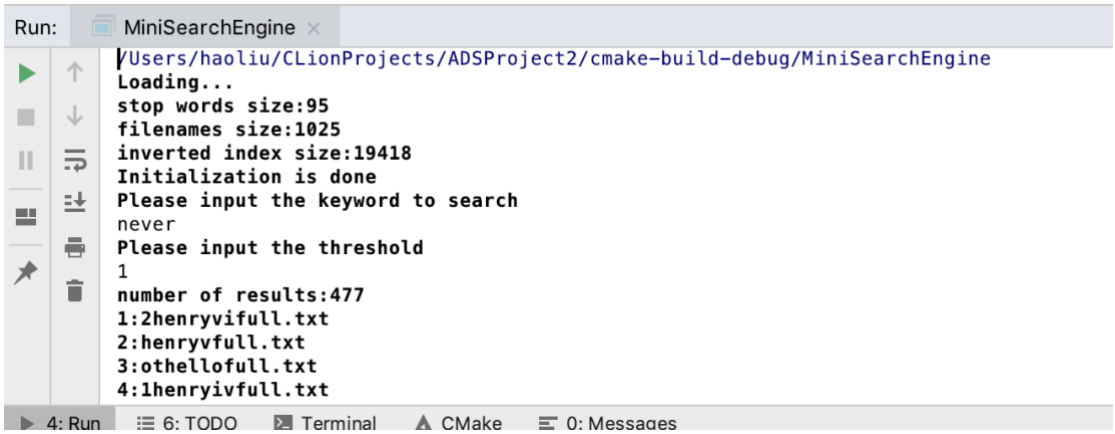


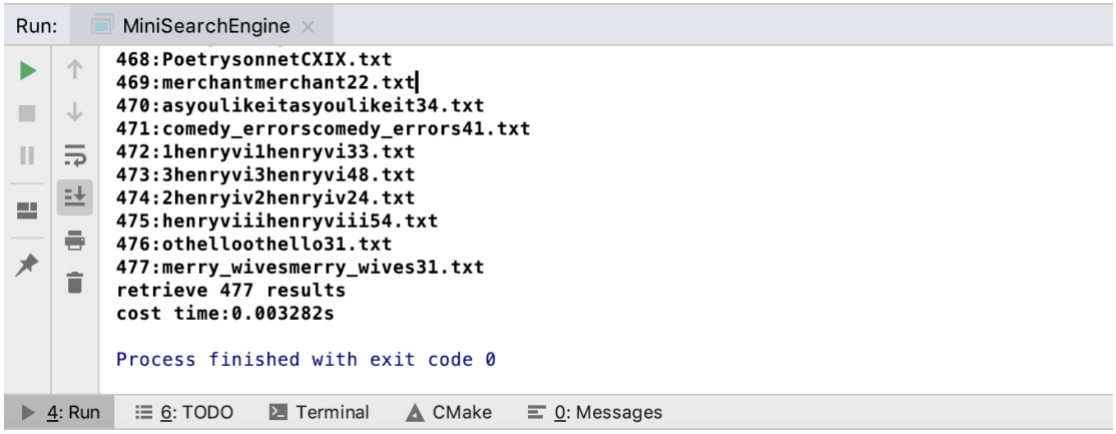


As we can see, the pretreater works very well.

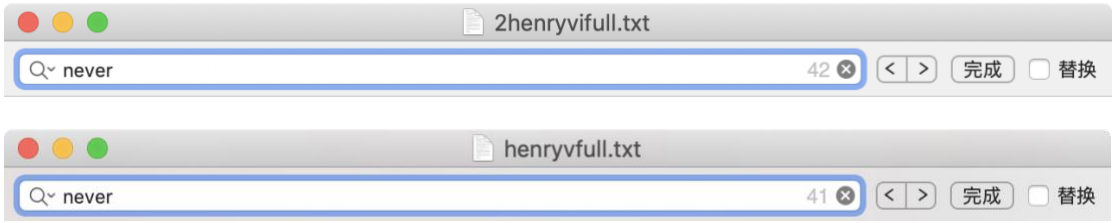
3.2 Search Engine

We use the keyword never to test our engine and the inverted index.



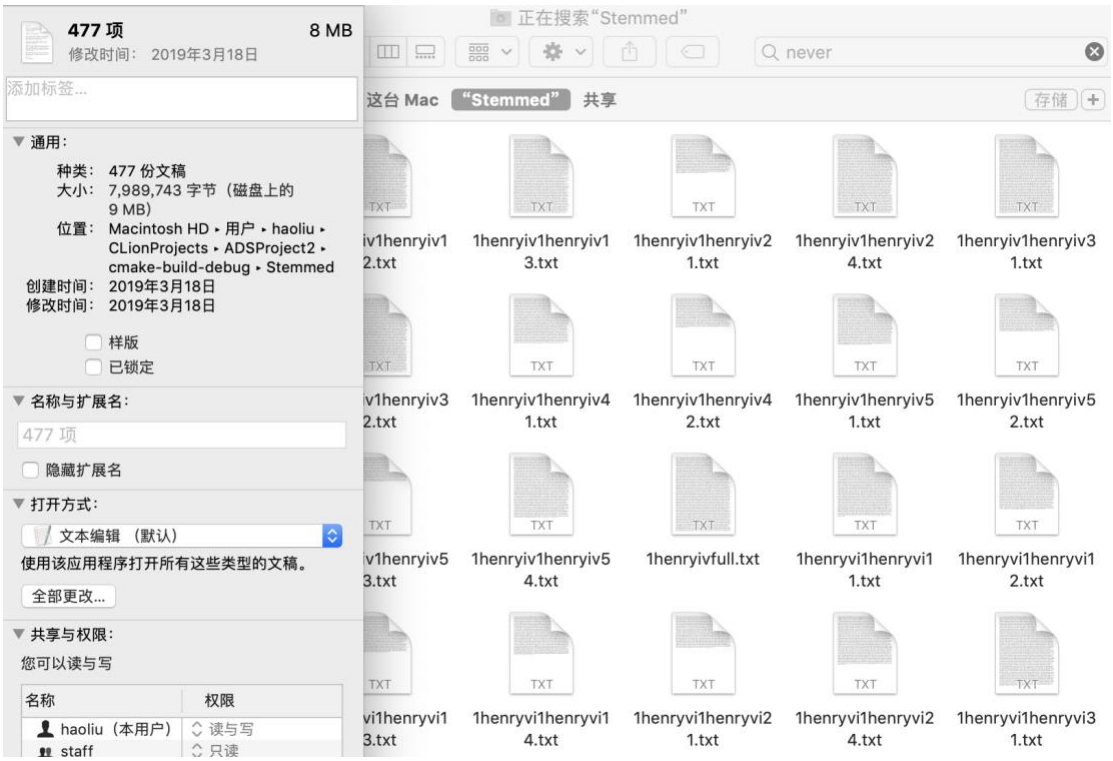


The results is ranked by the frequency of the keyword's appear.



The first file contains 42 keywords, the second file contains 41keywords.

Then we use the Mac OS's file search function to test the correctness of the results.



As we can see, there are 477 files that contain the keyword never, which fits our results very well.

If we change the threshold to 0.5.

```
Run:  MiniSearchEngine
/Users/haoliu/CLionProjects/ADSProject2/cmake-build-debug/MiniSearchEngine
Loading...
stop words size:95
filenames size:1025
inverted index size:19418
Initialization is done
Please input the keyword to search
never
Please input the threshold
0.5
```

```
Run:  MiniSearchEngine
230:henryviiihenryviii12.txt
231:3henryvi3henryvi43.txt
232:llllll21.txt
233:1henryvi1henryvi12.txt
234:2henryvi2henryvi410.txt
235:1henryvi1henryvi21.txt
236:winters_talewinters_tale53.txt
237:1henryiv1henryiv53.txt
238:much_adomuch_ado41.txt
239:richardiirichardii34.txt
retrieve 239 results
cost time:0.001531s

Process finished with exit code 0
```

The number of results will be half of the former case, just like what we

expect.

Chapter 4: Analysis and Comments

The map and set of C++ STL are implemented by RBT.

4.1 Time complexity:

Pretreater:O(N)

Read and stem words O(N)

void loadStopWords():O(N)

Read file O(N)

Insert to index O(logN)

void loadFilenames():O(N)

Read file O(N)

Insert to map O(logN)

void initialIndex():O(N)

Read file O(N)

Insert in map O(logN)

vector<string> search(string& keyWords, double threshold = 1.0):O(N)

Find in map O(logN)

Sort in vector O(logN)

Run over the elements in vector O(N)

4.2 Space complexity:

Pretreater:O(N)

Using stream to deal with the files requires no extra space.

But sort the words by their frequency require O(N) space.

void loadStopWords():O(1)

Simple reading from file, require no extra space.

void loadFilenames():O(1)

Simple reading from file, require no extra space.

void initialIndex():O(1)

No requirement for extra space.

vector<string> search(string& keyWords, double threshold = 1.0):O(N)

Using a vector to sort requires O(N) extra space.

4.3 Comments

The efficiency of our search engine can be improved. We use many C++ STL containner in our project, which are good enough for a mini search engine. If we design our own data structure like hash_map, maybe the process of index building and search can be speeded up.

Appendix: Source Code

Pretreater

```
 1 #include <iostream>
 2 #include <ratio>
 3 #include <fstream>
 4 #include <string>
 5 #include <map>
 6 #include <vector>
 7 #include <string>
 8 #include <set>
 9 #include <dirent.h>
10 #include <algorithm>
11 #include "stemmer/porter2_stemmer.h"
```

```
12
13 using namespace std;
14
15 bool compareFreq(pair<string, int> &a, pair<string, int>
&b)
16 {
17     return (a.second > b.second);
18 }
19
20 int main()
21 {
22     set<string> stopWords;
23     vector<string> documents;
24     string stopWordsFile("StopWordsList.txt");
25     string filenamesFile("FilenamesList.txt");
26     ifstream stopWordsIn(stopWordsFile);
27     ifstream FilenamesIn(filenamesFile);
28     double duration;
29     clock_t t1, t2;
30
31     string oDir = "ShakespeareComplete"; //原文文件夹名
32     string dDir = "Stemmed";  //目标文件夹名
33
34
35     t1 = clock();
36     system("mkdir Stemmed");
37
38     DIR *dir;
39     struct dirent *entry;
40     dir = opendir(oDir.c_str());
41
42     ifstream in;
43     ofstream out;
44     string filename;
45     string currentWord;
```

```
46      map<string, int> termFreq;
47      int docNum = 0;
48      ofstream filenamesOut(filenamesFile);
49      while((entry = readdir(dir)) != NULL){
50          filename = entry->d_name;
51          if(filename == "." || filename == "..") continue;
52          in.open(oDir + "/" + filename);
53          out.open(dDir + "/" + filename);
54
55          while (in >> currentWord)
56          {
57              Porter2Stemmer::trim(currentWord);
58              Porter2Stemmer::stem(currentWord);
59
60              if(currentWord == "") continue;
61              termFreq[currentWord]++;
62
63              out<<currentWord<<' ';
64          }
65          in.close();
66          out.close();
67
68          documents.push_back(filename);
69          filenamesOut<<filename<<' ';
70
71          docNum++;
72      }
73
74      typedef pair<string, int> pair;
75      vector<std::pair<string, int>> vecTf;
76      copy(termFreq.begin(), termFreq.end(),
back_inserter<vector<pair>>(vecTf));//复制到vector 容器中
77      sort(vecTf.begin(), vecTf.end(), compareFreq);
78      ofstream StopWordOut(stopWordsFile);
79      for(auto it = vecTf.begin(); it != vecTf.end(); ++it){
```

```
80          if(it->second > 2800)//出现次数大于2800的认定为stop
words，试出来比较合适
81          {
82              stopWords.insert(it->first);
83              StopWordOut << it->first << ' ';
84          }
85          else break;
86      }
87      cout << "Pretreatment is done."<<endl;
88
89      t2 = clock();
90      duration = double(t2-t1)/CLOCKS_PER_SEC;
91      cout<<"cost time:"<<duration<<"s"<<endl;
92      return 0;
93  }
```

main.cpp

```
 1 #include <iostream>
 2 #include <ratio>
 3 #include "SearchEngine.h"
 4 int main() {
 5     clock_t t1, t2;
 6     double duration;
 7     string keyWord;
 8     searchEngine searchShakespeare;
 9     cout << "Loading..." << endl;
10     searchShakespeare.loadStopWords();
11     searchShakespeare.loadFilenames();
12     searchShakespeare.initialIndex();
13     cout << "Initialization is done" << endl;
14     cout << "Please input the keyword to search" << endl;
15
16     float threshold;
17     cin >> keyWord;
18     cout << "Please input the threshold" << endl;
19     cin >> threshold;
```

```
    t1 = clock();
    auto const &result = searchShakespeare.search(keyWord);
    int numOfResult = int(result.size()*threshold) + 1;
    if (numOfResult == 0)cout << "Retrieve no result, you may input a stop word. Please try again." << endl;
    else{
        for (int i = 0; i < numOfResult; i++) {//保证如果有结果，至少返回一个结果
                cout << i + 1 << ":" << result[i] << endl;
        }
    }
    t2 = clock();
    duration = double(t2-t1)/CLOCKS_PER_SEC;
    cout<<"retrieve "<<numOfResult<<" results"<<endl;
    cout<<"cost time:"<<duration<<"s"<<endl;
    return 0;
}
```

SearchEngine.h

```
//
// Created by Hao Liu on 2019-03-19.
//

#ifndef MINISEARCHENGINE_SEARCHENGINE_H
#define MINISEARCHENGINE_SEARCHENGINE_H

#include <string>
#include <map>
#include <set>
#include <vector>

using namespace std;

typedef vector<pair<string, int>> postList;//文件名，出现次数

class searchEngine{
```

```
18 private:
19     set<string> stopWords;//包含所有 stop words 的集合
20     vector<string> filenames;
21     string dirname = "Stemmed";//存放文本的子目录名
22     map<string, postList*> invertedIndex;//搜索时使用的倒排文
件索引
23
24 public:
25     void loadStopWords();//加载 stop words 到集合中
26     void loadFilenames();//加载文本文件名到 vector 中
27     void initialIndex();//初始化倒排文件索引
28     vector<string> search(string& keyWords, double threshold
= 1.0);//搜索，输入关键词和 threshold，返回包含关键词的文件名列表
29 };
30 #endif //MINISEARCHENGINE_SEARCHENGINE_H
```

SearchEngine.cpp

```
1 //
2 // Created by Hao Liu on 2019-03-19.
3 //
4 #include <iostream>
5 #include <vector>
6 #include <string>
7 #include <fstream>
8 #include <sstream>
9 #include <algorithm>
10 #include "stemmer/porter2_stemmer.h"
11 #include "SearchEngine.h"
12
13 using namespace std;
14
15 void searchEngine::loadStopWords() {
16     string filename("StopWordsList.txt");
17     ifstream stopWordsStream(filename);
18     if(!stopWordsStream.is_open()){
19         cout<<"can't open stop words list."<<endl;
```

```
20          return;
21      }
22      string word;
23      while(stopWordsStream>>word){
24          stopWords.insert(word);
25      }
26      cout<<"stop words size:"<<stopWords.size()<<endl;
27  }
28
29  void searchEngine::loadFilenames() {
30      string filename("FilenamesList.txt");
31      ifstream filenamesStream(filename);
32      if(!filenamesStream.is_open()){
33          cout<<"can't open filenames List"<<endl;
34          return;
35      }
36      string name;
37      while(filenamesStream>>name){
38          filenames.push_back(name);
39      }
40      cout<<"filenames size:"<<filenames.size()<<endl;
41  }
42
43  void searchEngine::initialIndex() {
44      string currentWord;
45      for(const auto &filename : filenames){
46          ifstream inStream(dirname+"/"+filename);
47          if(!inStream.is_open())cout<<"Open file
fail."<<endl;
48          while(inStream>>currentWord){
49              if(stopWords.find(currentWord) ==
stopWords.end()){//单词不在stop
50                  auto iter = invertedIndex.find(currentWord);
51                  if(iter == invertedIndex.end()) {//词不在索引
中
```

```
52                     auto *tmp = new postList;
53                     tmp->push_back(pair<string,
int>(filename, 1));
54                     invertedIndex.insert(pair<string,
postList*>(currentWord, tmp));
55                     //cout<<currentWord<<endl;
56                 }
57                 else{//如果该词已经在索引中
58                     if(iter->second->back().first ==
filename){
59                         iter->second->back().second ++;
60                     }
61                     else{
62                         iter->second->push_back(pair<string,
int>(filename, 1));
63                     }
64                 }
65             }
66         }
67     }
68     cout<<"inverted index
size:"<<invertedIndex.size()<<endl;
69 }
70 bool compareFreq(pair<string, int>& a, pair<string, int>&
b){
71     return a.second > b.second;
72 }
73 vector<string> searchEngine::search(string& keyWords,
double threshold) {//返回包含所有文件名的vector
74     stringstream words;
75     string currentWord;//当前要搜索的单词
76     vector<string> filenames;
77     currentWord = keyWords;
78     Porter2Stemmer::trim(currentWord);
79     Porter2Stemmer::stem(currentWord);//处理一下关键词
```

```
80     auto list = invertedIndex.find(currentWord);
81     if(list != invertedIndex.end()) {//有结果
82         sort(list->second->begin(), list->second->end(),
compareFreq);//把文件名按关键词出现次数排序
83         for (auto tmp : *(list->second)) {
84             filenames.push_back(tmp.first);
85             //cout<<filenames.size();
86         }
87     }
88     cout<<"number of all results:"<<filenames.size()<<endl;
89     return filenames;
90 }
```

References

[1] Mark Allen Weiss, “data structure and algorithm analysis”, *Addison Wesley*, 1996

[2] Smassung, https://github.com/smassung/porter2_stemmer

[3] Haoxiong, https://github.com/liuhaoxiong/

[4] Roy Binux, https://github.com/binux/pyspider

Author List

Programming : Hao Liu

Testing: Hao Liu

Documentation: Hao Liu

Declaration

We hereby declare that all the work done in this project titled "Binary Search Tree" is of our independent effort as a group.

Signatures

刘皓

燕帅

任启岚